Cook'em Cookalicious

Team 5: James Kizer & Sean Herman

GitHub: Cornell-CS5356-Fall2014/Team-5
Webapp: dry-beach-2224.herokuapp.com

Introduction

Cook'em Cookalicious (CC) is a food journaling web and iOS application. The iOS application is focused on content creation, while the webapp is dedicated to a read-only experience. Users may create journal entries describing their cooking creations, with photos captured from their mobile device included in updates. Users may also supplement journal entries with rich recipe data from the 3rd party Yummly API.

Full Stack description

Back-end

CC is powered by Node.js and Express.js on the backend. These 2 tools serve all the API endpoints for the product across the iOS and webapp. All content is fetched and inserted via these same common Express API routes (e.g., /users, /journalentries, /photos). We authenticate users on the site using either standard username/password, or with the user's existing Facebook account, by way of Facebook's OAuth library. The Passport framework for Express was used to integrate Facebook login into the backend server.

To process photo files posted to the /photos endpoint, we use the multiparty streaming form-data parser. Thumbnail images are created on the server using the gm javascript library, underpinned by ImageMagick.

We use MongoDB to store user data on our server, hosted at MongoHQ (AKA Compose.io). Our Node/Express API is hosted on Heroku, and is accessible at dry-beach-2224.herokuapp.com.

Front-end

Our web front-end is powered by AngularJS supplemented with Bootstrap. The Angular components are all wrappers around the underlying Express/Node backend API. Our mobile application was developed in iOS, with a handful of additional frameworks and libraries, including AFNetworking, Facebook OAuth, iCarousel for photo display, and CHTCollectionViewWaterfallLayout for journal entry display.

Deep Dives

Social

We implemented some rudimentary social features into the CC experience. On the iOS app users may follow and unfollow other members of the site. On the webapp, a list of users can be viewed, including following counts, and detailed views of followers and following.

Recipe API

We expose a recipe API on our own server that wraps Yummly's own recipe API. This allows users of our webapp and mobile application to search for recipes, and include these Yummly recipes in CC journal entries. On the webapp, users may browse listings of recipes, given some query string, and view the details of a particular recipe. Outbound links point to the original source of the recipe, as reported by Yummly. When including a Yummly recipe in a journal entry, we only save the Yummly recipe Id inside a post. With this data alone, we can query our wrapper on Yummly's API to get the full details for a recipe when needed.

Facebook Authentication

In addition to a recipe API, we also implemented product authentication using Facebook's OAuth offering.

Interaction sketch between back and front end

POSTing Journal Entries from iOS

On the mobile iOS application, users begin a journal entry by clicking New Entry from the Food Feed view. At this point, existing picture from their phone's photo library, or photos captured directly from the device may be added to the entry.

When a user chooses to add a Recipe, the mobile application queries our web server's Yummly API wrapper at hostname/recipes/q=<query_string>. The server returns a JSON object, which is then rendered in a list view in the app. Individual recipes from this listview may be selected, with the details and photos visible. These detail views, when selected, will query perform a GET request to our webserver at /recipe/<recipe_id>. Again, this endpoint returns a JSON object, which is de-serialized, and rendered within the application.

Finally, the user adds a title and detail text to the journal entry, and presses Post. At this point, the iOS app performs a POST request to our webserver at hostname/journalentries. The server catches this request, uploads any attached photos, and inserts a new journal entry into the MongoDB database. The photos POSTed to /photos automatically trigger the normal thumbnailing process, which occurs asynchronously, in order to avoid requiring the user to wait for this costly operation to complete before responding.

Pivots

We met most of the minimum entry requirements without too much difficulty, aside from the photo resizing. We spent a full weekend working out different approaches to getting image resizing working properly. Eventually, we wound up using the javascript gm library, backed by ImageMagick simply because the software is already installed by default on our host, Heroku. Simple callback functions are used, rather than more sophisticated Promises, to ensure the thumbnailing process occurs asynchronously.

We did have a fair amount of trouble settling on a deep dive. We wound up settling on some basic Social mechanisms, and additional 3rd party API integration. The amount of time required to get the MER up and running did force us to scale back some originally lofty ambitions.

Conclusion

We both learned a great deal from this project in a very short period of time. Before working on CC, neither of us had worked with Javascript in any form. We chose a MEAN (Mongo, Express, Angular, Node) stack largely in an effort to gain some exposure to Javascript. In this respect, the project was a respect.

Naturally, choosing a new language did result in some disappointing delays, as just about every feature required a great deal of iteration. Had we worked with a language we both knew better, we might have completed more features, or delivered a more polished product, but still, we're both generally happy with the end result, and with the class overall.